

(19)日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平9-91143

(43)公開日 平成9年(1997)4月4日

(51)Int.Cl. ⁶	識別記号	府内整理番号	F I	技術表示箇所
G 06 F 9/44	5 3 0		G 06 F 9/44	5 3 0 J
9/445			13/00	3 5 1 H
13/00	3 5 1			3 5 7 Z
	3 5 7		9/06	4 2 0 J

審査請求 未請求 請求項の数15 O L (全20頁)

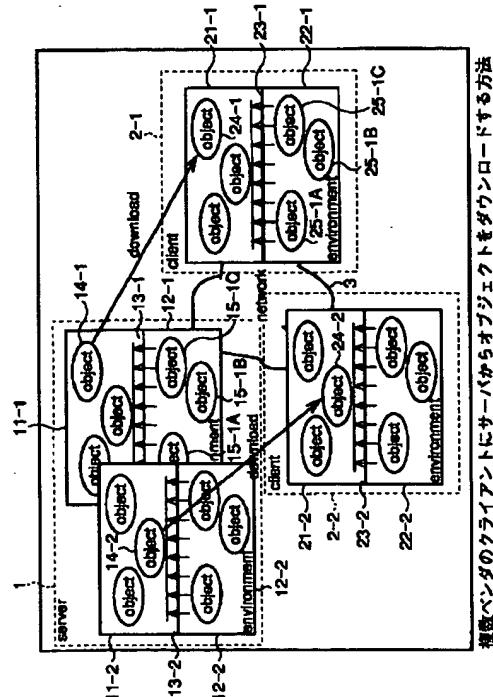
(21)出願番号	特願平8-67881	(71)出願人	000002185 ソニー株式会社 東京都品川区北品川6丁目7番35号
(22)出願日	平成8年(1996)3月25日	(72)発明者	横手 靖彦 東京都品川区北品川6丁目7番35号 ソニ ー株式会社内
(31)優先権主張番号	特願平7-178625	(74)代理人	弁理士 稲本 義雄
(32)優先日	平7(1995)7月14日		
(33)優先権主張国	日本 (JP)		

(54)【発明の名称】 データ処理方法および装置

(57)【要約】

【課題】 クライアントの構成を簡略化し、低コスト化を可能にする。

【解決手段】 サーバ1とクライアント2-1, 2-2をネットワーク3を介して接続する。サーバ1には、クライアント2-1, 2-2が有する実行環境22-1, 22-2に対応する実行環境12-1, 12-2を予め用意するようする。クライアント2-1, 2-2のアプリケーションプログラム21-1, 21-2において、所定のオブジェクト24-1, 24-2が必要になったとき、サーバ1のアプリケーションプログラム11-1, 11-2のオブジェクト14-1, 14-2をダウンロードする。



【特許請求の範囲】

【請求項1】 複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成される、前記アプリケーションプログラムの動作を規定する実行環境と、前記アプリケーションプログラムと前記実行環境の間のインターフェースを規定するアプリケーションプログラムインターフェースとを備えるサーバと、前記サーバより前記アプリケーションプログラムをダウンロードするクライアントと、

を備えるデータ処理システムにおけるデータ処理方法において、

前記サーバは、前記クライアントに前記アプリケーションプログラムをダウンロードするとき、前記クライアントが、ダウンロードする前記アプリケーションプログラムの実行環境を有するか否かを検査し、その検査結果に対応して前記アプリケーションプログラムを前記クライアントにダウンロードすることを特徴とするデータ処理方法。

【請求項2】 前記クライアントが、ダウンロードする前記アプリケーションプログラムの実行環境を有しないとき、前記サーバから前記クライアントにオブジェクトをダウンロードし、前記サーバにおける前記実行環境と同一の実行環境を前記クライアントに構築し、その後、前記サーバから前記クライアントに前記アプリケーションプログラムをダウンロードすることを特徴とする請求項1に記載のデータ処理方法。

【請求項3】 前記アプリケーションプログラムのオブジェクトは、並行オブジェクトであることを特徴とする請求項1に記載のデータ処理方法。

【請求項4】 前記サーバは、前記アプリケーションプログラムの実行に必要な前記実行環境のオブジェクトをインクリメンタルにダウンロードすることを特徴とする請求項1に記載のデータ処理方法。

【請求項5】 前記クライアントの実行環境には、最低限の標準的機能として、
前記サーバから前記クライアントへオブジェクトをダウンロードするダウンロード機能と、
前記実行環境の互換性を検査する検査機能と、
前記実行環境を構築する構築機能とを具備させ、
必要に応じてその他の機能をダウンロードさせることを特徴とする請求項2に記載のデータ処理方法。

【請求項6】 前記クライアントには、最低限の標準的機能として、

デバイスドライバのための実行環境と、
システムオブジェクトのための実行環境と、
実行環境のための実行環境とを具備させ、
必要に応じてその他の機能をダウンロードさせることを特徴とする請求項2に記載のデータ処理方法。

【請求項7】 前記デバイスドライバは、インプットドライバ、タイマドライバ、スクリーンドライバを含み、

前記システムオブジェクトは、インプットハンドラ、ポートプロトコル、メモリマネージャの各オブジェクトを含むことを特徴とする請求項6に記載のデータ処理方法。

【請求項8】 前記サーバと前記クライアントは、前記クライアントの前記実行環境の互換性を検査するため、前記オブジェクトと前記実行環境について記述したデスクリプションを含むフィーチャストラクチャを授受することを特徴とする請求項1に記載のデータ処理方法。

【請求項9】 複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成される、前記アプリケーションプログラムの動作環境を規定する実行環境と、前記アプリケーションプログラムと前記実行環境の間のインターフェースを規定するアプリケーションプログラムインターフェースとを備え、クライアントにアプリケーションプログラムをダウンロードさせるデータ処理装置において、

前記クライアントに前記アプリケーションプログラムをダウンロードするとき、前記クライアントが、ダウンロードする前記アプリケーションプログラムの実行環境を有するか否かを検査する検査手段と、
前記検査手段の検査結果に対応して前記アプリケーションプログラムを前記クライアントにダウンロードするダウンロード手段とを備えることを特徴とするデータ処理装置。

【請求項10】 複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成される、前記アプリケーションプログラムの動作環境を規定する実行環境と、前記実行環境の間のインターフェースを規定するアプリケーションプログラムインターフェースとを備え、サーバからアプリケーションプログラムをダウンロードするデータ処理装置において、

前記サーバから前記アプリケーションプログラムをダウンロードするとき、ダウンロードする前記アプリケーションプログラムの実行環境に関する告知を行う告知手段と、

前記告知手段の告知に対応して前記サーバから前記アプリケーションプログラムをダウンロードするダウンロード手段とを備えることを特徴とするデータ処理装置。

【請求項11】 複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成され、前記アプリケーションプログラムに対して実行環境を提供するシステムオブジェクトからなるデータ処理装置において、

中間コードに変換された前記アプリケーションプログラムを解釈し、実行する第1の実行手段と、
前記中間コードを動的にコンパイルし、バイナリコードを生成するバイナリコード生成手段と、

前記バイナリコードおよび前記システムオブジェクトを実行する第2の実行手段とを備えることを特徴とするデータ処理装置。

【請求項12】前記第1の実行手段は、前記第2の実行手段を介して前記システムオブジェクトを実行することを特徴とする請求項11に記載のデータ処理装置。

【請求項13】前記アプリケーションプログラムの実行状態を示す第1の構造体と、

前記システムオブジェクトが提供する実行環境のアプリケーションプログラムインターフェースを決定する第2の構造体とをさらに備え、

前記第1の実行手段および前記第2の実行手段は、前記第1の構造体と前記第2の構造体に基づいて、前記アプリケーションプログラムの実行を制御することを特徴とする請求項11に記載のデータ処理装置。

【請求項14】前記第1の実行手段は、前記オブジェクトの所定のものにより発行され、所定のオペランドで指定された所定のオペレーションを実行する第1の命令と、前記オペレーションを実行した後、前記命令を発行した前記オブジェクトに制御を戻す第2の命令とを有することを特徴とする請求項11に記載のデータ処理装置。

【請求項15】複数のオブジェクトで構成されるアプリケーションプログラムと、複数のオブジェクトで構成され、前記アプリケーションプログラムに対して実行環境を提供するシステムオブジェクトからなるデータ処理装置におけるデータ処理方法において、

中間コードに変換された前記アプリケーションプログラムを解釈し、実行する方法と、前記中間コードを動的にコンパイルし、生成されたバイナリコードを実行する方法とにより、前記アプリケーションプログラムを実行することを特徴とするデータ処理方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明はデータ処理方法および装置に関し、特に構成を簡略化し、低コスト化することができる様にした、データ処理方法および装置に関する。

【0002】

【従来の技術】最近、パーソナルコンピュータが普及してきた。このパーソナルコンピュータでネットワークを介して所定のサーバにアクセスし、所定の情報を得ることができる。

【0003】このようなパーソナルコンピュータにおいて各種の処理を行うには、アプリケーションプログラムが必要となる。そこで、各ユーザは、そのパーソナルコンピュータのOSに合ったアプリケーションプログラムを購入し、インストールして用いるようにしている。

【0004】

【発明が解決しようとする課題】このように、OSが異

なると、アプリケーションプログラムも異なるものとなるため、各ユーザは、自分のOSに合ったアプリケーションプログラムを選択して購入する必要がある。また、アプリケーションプログラムを提供する側（アプリケーションプログラムを設計する側）においても、実質的に同一の処理を行うアプリケーションプログラムを複数（OSの数だけ）設計し、用意しなければならず、多くの労力が必要になると同時に、コスト高となる課題があった。

【0005】また、同様のことが、同一のOSにおける各アプリケーションプログラムにおいても発生していた。すなわち、1つのアプリケーションプログラムと、それとは異なる他のアプリケーションプログラムとが、同一のOS上において動作するものであったとしても、2つのアプリケーションプログラムはそれぞれ別個に設計しなければならず、結果的に、1つのアプリケーションプログラムを提供するのに必要な労力とコストが高くなる課題があった。

【0006】本発明はこのような状況に鑑みてなされたものであり、1つのアプリケーションプログラムを、簡単かつ低コストで提供できるようにするものである。

【0007】

【課題を解決するための手段】請求項1に記載のデータ処理方法は、サーバが、クライアントにアプリケーションプログラムをダウンロードするとき、クライアントが、ダウンロードするアプリケーションプログラムの実行環境を有するか否かを検査し、その検査結果に対応してアプリケーションプログラムをクライアントにダウンロードすることを特徴とする。

【0008】請求項9に記載のデータ処理装置は、クライアントにアプリケーションプログラムをダウンロードするとき、クライアントが、ダウンロードするアプリケーションプログラムの実行環境を有するか否かを検査する検査手段と、検査手段の検査結果に対応してアプリケーションプログラムをクライアントにダウンロードするダウンロード手段とを備えることを特徴とする。

【0009】請求項10に記載のデータ処理装置は、サーバからアプリケーションプログラムをダウンロードするとき、ダウンロードするアプリケーションプログラムの実行環境に関する告知を行う告知手段と、告知手段の告知に対応してサーバからアプリケーションプログラムをダウンロードするダウンロード手段とを備えることを特徴とする。

【0010】請求項11に記載のデータ処理装置は、中間コードに変換されたアプリケーションプログラムを解釈し、実行する第1の実行手段と、中間コードを動的にコンパイルし、バイナリコードを生成するバイナリコード生成手段と、バイナリコードおよびシステムオブジェクトを実行する第2の実行手段とを備えることを特徴とする。

【0011】請求項15に記載のデータ処理方法は、中間コードに変換されたアプリケーションプログラムを解釈し、実行する第1の方法と、中間コードを動的にコンパイルし、生成されたバイナリコードを実行する第2の方法とにより、アプリケーションプログラムを実行することを特徴とする。

【0012】請求項1に記載のデータ処理方法においては、クライアントにアプリケーションプログラムをダウンロードするとき、クライアントが、ダウンロードするアプリケーションプログラムの実行環境を有するか否かをクライアントとサーバ間で検査し、その検査結果に対応してアプリケーションプログラムをクライアントにダウンロードする。

【0013】請求項9に記載のデータ処理装置においては、検査手段が、クライアントにアプリケーションプログラムをダウンロードするとき、クライアントが、ダウンロードするアプリケーションプログラムの実行環境を有するか否かを検査し、ダウンロード手段が、検査手段の検査結果に対応してアプリケーションプログラムをクライアントにダウンロードする。

【0014】請求項10に記載のデータ処理装置においては、告知手段が、サーバからアプリケーションプログラムをダウンロードするとき、ダウンロードするアプリケーションプログラムの実行環境に関する告知を行い、ダウンロード手段が、告知手段の告知に対応してサーバからアプリケーションプログラムをダウンロードする。

【0015】請求項11に記載のデータ処理装置においては、第1の実行手段が、中間コードに変換されたプログラムを解釈し、実行し、バイナリコード生成手段が、中間コードを動的にコンパイルし、バイナリコードを生成し、第2の実行手段が、バイナリコードを実行する。例えば、動的コンパイルが困難な場合、中間コードを逐次解釈し、実行することができる。

【0016】請求項15に記載のデータ処理方法においては、中間コードに変換されたプログラムを解釈し、実行する第1の方法と、中間コードを動的にコンパイルし、生成されたバイナリコードを実行する第2の方法とにより、アプリケーションプログラムを実行する。例えば、動的コンパイルが困難な場合、中間コードを逐次解釈し、実行することができる。

【0017】

【発明の実施の形態】本発明のデータ処理方法を適用するシステム構成例を図1に示す。システムはサーバ1(データ処理装置)、クライアント2(データ処理装置)、ネットワーク3から構成されている。

【0018】すなわち、この実施例においては、サーバ1は2つのアプリケーションプログラムを有し、一方のアプリケーションプログラム11-1は、それを実行する環境を規定する実行環境12-1と、アプリケーションプログラム11-1と実行環境12-1との間のイン

タフェースを構成するアプリケーションプログラムインターフェース(AP1)13-1を有している。

【0019】アプリケーションプログラム11-1は、複数のオブジェクト14-1により構成され、また、実行環境12-1も、複数のオブジェクト15-1により構成されている。

【0020】同様に、アプリケーションプログラム11-2は、その環境を規定する実行環境12-2と、アプリケーションプログラム11-2と実行環境12-2との間のインターフェースとして機能するAP113-2を有している。

【0021】このアプリケーションプログラム11-2も、複数のオブジェクト14-2により構成され、また、実行環境12-2も複数のオブジェクト15-2により構成されている。

【0022】同様に、クライアント2も2つのアプリケーションプログラムを有し、一方のアプリケーションプログラム21-1は、その環境を規定する実行環境22-1と、アプリケーションプログラム21-1と実行環境22-1との間のインターフェースであるAP123-1を有している。アプリケーションプログラム21-1と実行環境22-1は、それぞれ複数のオブジェクト24-1と25-1により構成されている。

【0023】同様に、アプリケーションプログラム21-2も、その環境を規定する実行環境22-2とAP123-2を有し、アプリケーションプログラム21-2と実行環境22-2は、それぞれ複数のオブジェクト24-2と25-2により構成されている。

【0024】ここで、各オブジェクトはすべて、他のオブジェクトと並行して処理を実行する並行オブジェクトとして定義されている。1つのAP1の集合は1つの実行環境によって与えられるので、サーバ1、クライアント2中には、複数のAP1が存在することになる。

【0025】図1に示すように、また図2に拡大して示すように、アプリケーションプログラム11を複数のオブジェクト14の集まりで構成する。また、オブジェクト14を並行オブジェクトとして構成することにより、アプリケーションプログラム11は並行処理されることになり、実行速度の向上に貢献する。また、オブジェクト14は置き換える単位であるので、動作にバグのあるオブジェクト、性能上問題のあるオブジェクトなどを、誤りのないオブジェクトで置き換えることによって、アプリケーションプログラム11全体を作り替えることなく、問題点を解決できる。さらに、オブジェクト14を部品とし、既存のアプリケーションプログラムの部品としてのオブジェクトを組み合わせることによって、簡単に新しいアプリケーションプログラムを作ることができる。

【0026】ここで、アプリケーションプログラムとは、1つのサービス単位である。例えば、サーバ1から

の映像データを単に表示しているアプリケーションプログラム、VCR機能を用いて映像データを検索しているアプリケーションプログラム、メニューによってサービスを選択しているアプリケーションプログラム、ホームショッピングのアプリケーションプログラム、ホームショッピングと連結した家計簿アプリケーションプログラム、税金計算アプリケーションプログラム等である。

【0027】アプリケーションプログラム間でオブジェクトを共有することによって、操作性に共通点を持たせることができ。例えば、家計簿でデータを入力しているエディタと、ホームショッピングでのデータ入力エディタを共通化することができる。

【0028】次に並行オブジェクト (concurrent object) について説明する。並行オブジェクトの構成を図3に示す。並行オブジェクトであるオブジェクト14は、外部に公開されたメソッドエントリのテーブル14A、メソッドの本体14B、オブジェクトの状態を保持するメモリ領域14C、メソッドを実行する単一のスレッド14Dを有している。並行オブジェクトには1つの実行コンテキスト (スレッドと呼んでも良い) のみが存在する。従って並行オブジェクトは、1つのメッセージを受信して、その処理中には、新たに到着したメッセージの処理は、現在の実行が終了するまで行わない。

【0029】このように、オブジェクト内にスレッドを1個だけ配置するようにすると、次の利点が得られる。

【0030】(1) 複数のアクティビティ間の同期を気にする必要がない。すなわち、共有データが存在する場合に、セマフォア等の同期のための命令を用いて、共有データに対するアクセスを順序付けるといったを行う必要がなくなる。換言すれば、オブジェクトへのメッセージ送信が、その順序付けを含んでいることになる。

【0031】(2) そのため、同期の取り方のミスによるプログラム誤りが生じなくなると同時に、そのオブジェクトの再利用可能性が高まる。

【0032】(3) 例えばデバイスドライバを本方式で作成することによって、多くの場合に生じる、同期誤りを防ぐことができる。

【0033】(4) また、デバイスドライバの置き換えるによる同期誤りを防ぐことができるので、安全にデバイスドライバを置き換えることができる。

【0034】(5) デバイスドライバの、実際にハードウェアを制御する部分以外の部分を、OSと独立に作成することができる。これによって、従来から開発のかなりの時間を占めていたデバイスドライバを、共通に開発することができる。開発期間の短縮につながる。

【0035】(6) オブジェクト間の実行制御に関する記述をアプリケーションプログラムの記述から除くことができる。例えば、マルチスレッドを用いた手法では

(複数のスレッドを用いる場合には)、スレッドの実行制御をアプリケーションプログラム中にプログラムする必要があるために、スレッドのプログラミング環境が変更になると、アプリケーションプログラムを書き換える必要がある。しかしながら、スレッドが1個の場合には、アプリケーションプログラムにこの部分を記述する必要がないので、実行制御方法が変わってもアプリケーションプログラムを書き直す必要がない。並行オブジェクトのその実行環境への最適な実行制御方法は、システムが、後述するオブジェクトの動的拡張の原理を用いて提供する。

【0036】(7) 従って、アプリケーションプログラムを記述する場合には、並列処理を考える必要がない。並行オブジェクトが並列処理の単位であるので、並行オブジェクトをプログラムしていけば、後はシステムが自動的にそのハードウェアに最適な実行制御を行って並列処理が行われる。従来の手法では、いくつプロセスを生成するか、いくつスレッドを生成するか、といったことをプログラミング時に指定しなければならず、この指定は、ハードウェアの性能を考慮しないと、そのアプリケーションプログラムは特定のハードウェア専用のものになってしまうが、本方式によれば、そのようなことはない。

【0037】本システムにおいては、オブジェクトは必要に応じてダウンロードされる。複数ベンダのクライアント2に対してサーバ1からオブジェクトをダウンロードする場合のシステム例を図4に示す。サーバ1上には、それぞれのベンダ用のクライアントAPI13(13-1, 13-2)が、実行環境12(12-1, 12-2)によって実現されている。

【0038】オブジェクトをクライアント2(2-1, 2-2)にダウンロードするとき、クライアント2上に、サーバ1上の実行環境12と同じ実行環境22(22-1, 22-2)が存在するか否かを調べ、存在する場合には、オブジェクトをダウンロードする。存在しない場合には、クライアント2上にサーバ1上の実行環境と同一の実行環境を構築した後、ダウンロードする。

【0039】例えば、図4において、サーバ1のアプリケーションプログラム11-1のオブジェクト14-1を、クライアント2-1のアプリケーションプログラム21-1のオブジェクト24-1としてダウンロードする場合、クライアント2-1の実行環境22-1に、サーバ1の実行環境12-1のオブジェクト15-1Aに対応するオブジェクト25-1Aが必要であるとき、例えば、実行環境12-1のオブジェクト15-1B(検査手段)は、実行環境22-1のオブジェクト25-1B(告知手段)にフィーチャストラクチャ(後述する)を問い合わせる。そして、その回答に対応して、実行環境12-1のオブジェクト15-1C(ダウンロード手段)と実行環境22-1のオブジェクト25-1C(ダ

9

【0040】従来の手法では、ダウンロードするオブジェクトは、クライアントのAPIに合わせて作成する必要があった。例えば、クライアントがUNIXシステムの場合には、サーバ上では同じUNIXシステムを用いるか、あるいは何らかのクロス開発環境を構築してオブジェクトを作成する必要がある。もし、サーバとクライアントが同じ実行環境を備えなければいけない場合には、クライアント装置は一般に高価な計算資源を備える必要がある。例えば、専用の実行環境を備える場合と比べて、より多くのメモリを備える必要がある。また、十分な実行速度を保証するために、高速のCPU(Central Processing Unit)を備える必要がある。これは、装置のコスト増につながる。

【0041】これに対して、本システムでは、アプリケーションのダウンロードと同時に、その実行環境をもダウンロードすることで、この問題を解決する。すなわち、クライアント2で現在必要とする実行環境22のみをクライアント2に構築することによって、不必要的な資源をクライアント2に用意しなくてもすむようになる。例えば、クライアント2が3Dのグラフィックスを必要としない場合には、そのライブラリは必要なくなる。

【0-0-4-2】また、クライアント2がVOD-(V-i d e o O n D e m a n d)で映画を見ている場合には、ユーザとのインタラクションのためのサービス（映画を見るときは不要となるサービス）をクライアントから一時削除することによって、その分の計算資源を他の仕事に割り振ることができる。例えば、その資源を、サーバ1からの映像データのプリフェッч用のバッファに使うことができる。インタラクションのためのサービスは、それが必要になった時点でサーバ1からダウンロードされる。

【0043】本システムでダウンロードされるオブジェクトとしては次のものが考えられる。

【0044】(1) すべてのアプリケーションプログラム

【0045】(2) クライアントが備えるハードウェア資源を制御するためのデバイスドライバ群(例えば、MPEG ドライバ、ATM ドライバ、画像制御ドライバ等)

【0046】(3) アプリケーションプログラムに対してシステムサービスを提供するオブジェクト群（例えば、VCRコマンド管理、ストリーム管理、実時間スケジューラ、メモリ管理、ウィンドウ管理、ダウンロード制御、通信プロトコル管理、実行管理等）

【0047】これらを組み合わせて、アプリケーションプログラムに対して最適な実行環境をクライアント上に

構築する。

【0048】サーバ1は、映像データやアプリケーションプログラムを送出する装置であったり、クライアント2にネットワーク3を通して情報を送出する装置である。一方クライアント2は、サーバ1からの情報を処理する装置であり、常にネットワーク3と接続されている必要はない。実行環境はアプリケーションプログラム毎に与えることができるので、アプリケーションプログラム毎に最適な実行環境を用意することができる。

【0049】従来の手法では、システム構築時にアプリケーションの特性をあらかじめ見積もっておく必要があった。例えば、アプリケーションプログラムが映像データを扱う必要があるときには、そのためのシステムサービス、例えば、実時間スケジューリングや映像データを扱うためのVCRのようなユーザインタフェースを備えている必要がある。また、アプリケーションが3Dグラフィックスを用いているならば、そのためのライブラリを備えている必要がある。そのため、システムは肥大になる傾向があった。UNIXやWindows(商標)

20 がこの典型例であり、バージョンがあがるごとにシステムが必要とするメモリ量は多くなっていった。本システムでは、アプリケーションの実行のために必要最小限の機能のみを備えていればよく、従来システムのこの問題点を解決する。

【0050】アプリケーションプログラム11を複数のオブジェクトの集まりとして構成することにより、また、そのオブジェクトを並行オブジェクトとして実装することにより、オブジェクト単位で並行実行が可能になり、アプリケーションプログラムの実行と同時にオブジ
30 ェクトをダウンロードすることができる。この時、図5に示すように、アプリケーションプログラムの実行に必要なオブジェクトをインクリメンタルにダウンロードすることにより、ユーザからは、単一のアプリケーションプログラムのロードにかかる時間を隠すことができる。

【0051】例えば、図5に示すように、サーバ1のアプリケーションプログラム11のオブジェクト14-1-1乃至14-1-11を、クライアント2のアプリケーションプログラム21のオブジェクト24-1-1乃至24-1-11としてダウンロードする必要がある場合、ランダムに各オブジェクトをダウンロードするではなく、アプリケーションプログラム21を実行する上において、最初に必要なオブジェクト14-1-1乃至14-1-3を、オブジェクト24-1-1乃至24-1-3として先にダウンロードする。

【0052】アプリケーションプログラム21は、さしあたって、この3つのオブジェクトが存在すれば、起動可能であるため、その処理を開始する。そして、その処理が実行されている間に、残りのオブジェクト14-1-4乃至14-1-11をアプリケーションプログラム50 21のオブジェクト24-1-4乃至24-1-11と

して、第2乃至第4のダウンロードで順次ダウンロードする。この第2乃至第4のダウンロードにおいても、処理上、先に必要となるオブジェクトから順番にダウンロードする。

【0053】ユーザは、アプリケーションプログラム21がオブジェクト24-1-1乃至24-1-3がダウンロードされ、その処理が開始された時点において、既にアプリケーションプログラム21の処理が開始されているため、あたかもすべてのオブジェクトのダウンロードが完了したものと認識することができる。すなわち、ユーザは、11個のオブジェクトをダウンロードするのに必要な時間より短い、3個のオブジェクトをダウンロードするのに必要な時間だけを意識することになる。換言すれば、ユーザに対して、8個のオブジェクトをダウンロードするための時間を実質的に隠す（意識させないようにする）ことができる。

【0054】このことは、図4を参照して説明した（また、図10を参照して後述する）実行環境22の構築をクライアント2上に行う場合にも当てはまる。この場合は、実行環境22を構成するオブジェクトのうち、アプリケーションプログラムの実行に必要な部分のみのオブジェクトを先にダウンロードすることによって、実行環境のすべてのオブジェクトをダウンロードする時間をユーザから隠すことができる。この手法は、システムのブートにも応用できる。

【0055】ここで、インクリメンタルダウンロードとは、アプリケーションプログラムや実行環境を一度にダウンロードしないで、それを構成するオブジェクト単位、あるいは、その一部を必要に応じてダウンロードすることを意味する。従来のパソコン通信におけるアプリケーションプログラムのダウンロードの場合には、圧縮されたアプリケーションプログラムを一気にダウンロードするので、ダウンロードがすべて終わらないと、そのアプリケーションプログラムは利用することができない。また、例えば、今までのシステムのブートでは、システム全体を全部メモリに読み込んでから立ち上がる。UNIXのディスクレスワークステーションの場合には、サーバからすべてのOSをメモリに読み込んでからシステムが起動するので、読み込みがすべて終了するまで、システムを利用できない。しかしながらインクリメンタルダウンロードすれば、そのようなことがなくなる。

【0056】この手法は、サーバ1とクライアント2としてのSTB（S e t T o p B o x）に応用して次のような効果がある。まず、STBの電源を入れるとすぐに利用できるようになる。現在のパーソナルコンピュータのように、システムが立ち上がるまでいらいらして待つ必要が無くなる。STBは家庭用電気製品としての性格が強いので、システムが立ち上がるまでユーザを待たせるのは好ましくない。

【0057】STBの電源が入れられると、STBは最初に必要なオブジェクトをダウンロードして実行を始めようとする。ユーザの待ち時間は最初のこのオブジェクトのダウンロード時間のみである。典型的なオブジェクトのダウンロード時間は数ミリ秒から数十ミリ秒であるので、適当なユーザインターフェースを備えることにより、この時間は十分ユーザにとって無視できる時間となる。以降は、システムの立ち上げプロセスの進行に従って、必要なオブジェクトがシステムの立ち上げプロセスと並行にダウンロードされる。

【0058】また、STBは、そのコストの観点から、サーバのような豊富な計算資源が用意されていないので、複数のアプリケーションを同時に実行する場合にも制約が生じる。例えば、ナビゲーションアプリケーションによって、VODサービスを選択して、1つの映画を鑑賞することを考えた場合、映画の鑑賞が始またら、ナビゲーションアプリケーションが占有していた資源（メモリ）を解放して、映画鑑賞アプリケーションのために使うことができる。そして、再び、ナビゲーションアプリケーションが必要になった時点で、その資源（メモリを管理するオブジェクト）をダウンロードする。

【0059】ここで、「必要になった時点」とは、オブジェクトに対して何らかのメッセージが送られた時点とする。すなわち、一番最初にダウンロードしたオブジェクトが、別のオブジェクトにメッセージを送った時点で、その受信オブジェクトをダウンロードする。オブジェクトの依存、参照関係を利用することによって、次のメッセージを送るオブジェクトをあらかじめダウンロードしておくことができる。これを、アプリケーションの実行と並行に行うことにより、メッセージ通信時におけるダウンロードによる遅延を少なくすることができる。これにより、インクリメンタルダウンロードの有効性を高めることができる。

【0060】また、実行環境12、22もオブジェクト15、25の集合体であり、アプリケーションプログラム11、21のオブジェクト14、24と同等に操作可能であるので、アプリケーションプログラム11、21に特化したダウンロードの順序を制御するメタオブジェクトをオブジェクト15、25の1つとして用意する（例えば図4のオブジェクト25-1Cをメタオブジェクトとする）ことができる。これによって、特定のアプリケーションに適した、そのオブジェクトが利用するオブジェクトのダウンロード順を指定することができ、インクリメンタルダウンロードによるユーザの待ち時間を最小にできる。

【0061】サーバ1からクライアント2へのオブジェクトのダウンロードの機能、それに伴うオブジェクトの実行環境の互換性の検査機能、および実行環境の構築機能は、本システムを実現する上で重要なものであり、本システムを構成するすべての装置（クライアント2）が

最低限有すべき機能とする。本明細書ではこの機能をメタ標準と呼ぶ。このメタ標準により、OS等の実行環境のAPIは自由に拡張することができるようになり、最小限の標準化とその拡張によって、今後のあらゆるタイプのアプリケーションに対応することができるようになる。

【0062】例えば、図6に示すシステムでは、各サーバ1-1, 1-2、各クライアント2-1, 2-2では、独自のAPIを持ったOSが稼働している。すなわちクライアント2-1においては、実行環境2-2-1に対応して、アプリケーションプログラム2-1-1のためのAPI2-3-1 (API #1) が構成されている。また、クライアント2-2においては、実行環境2-2-2によりアプリケーションプログラム2-1-2のためのAPI2-3-2 (API #3) が形成されている。

【0063】このため、これらのクライアント2-1, 2-2に対してプログラムをダウンロードするサーバには、これらのAPIに対応するAPIが予め用意されている。この実施例においては、サーバ1-1において、実行環境1-2-1によりアプリケーションプログラム1-1-1のためのAPI1-3-1が形成されており、このAPI1-3-1は、クライアント2-1におけるAPI2-3-1 (API #1) に対応するAPI (API #1) とされている。

【0064】同様にサーバ1-2において、実行環境1-2-3によりアプリケーションプログラム1-1-3に対応するAPI1-3-3 (API #3) が形成されており、このAPI1-3-3がクライアント2-2のAPI2-3-2 (API #3) に対応している。

【0065】従って、サーバ1-1, 1-2と、クライアント2-1, 2-2に、このメタ標準に対応するオブジェクトとして、オブジェクト15-1A乃至15-1C, 15-3A乃至15-3C, 25-1A乃至25-1Cおよび25-2A乃至25-2Cを設けている。その結果、クライアント2-1または2-2に対しては、サーバ1-1または1-2から、メタ標準プロトコルに従って必要なオブジェクトを適宜ダウンロードさせることができる。

【0066】各クライアントにおけるOSを1つのOSに標準化しようとするのがこれまでのこの分野における傾向であった。しかしながら、このようなメタ標準を規定し、サーバ側においてのみ、各クライアントのAPIに対応するAPIを具備させることで、標準を決定する必要がなくなる。

【0067】OSの標準を1つに規定していないことによって、アプリケーションを始め、システムサービスを実現しているオブジェクトは、OSとは独立に構成することが可能になる。すなわち、ある実行環境用に書かれたソフトウェアを、マイグレーションによって別の実行環境用に自動的に再構成すればよいことになる。従来の

システムには、この機能は存在しない。例えば、UNIX用に書かれたソフトウェアはWindows上では、書き直さないと動作しない。アプリケーションレベルのソフトウェアで、この機能を実現するには、OS依存性を吸収するソフトウェアを導入する必要がある。しかし、デバイスドライバを始めとする、システムサービスを実現するオブジェクトのOSの独立性は、本手法を用いで可能になる。

【0068】10 このように、オブジェクトをダウンロードするようになると、クライアント2のオブジェクトは、図7に示すように、動的に変更することができる。すなわち、既存のオブジェクトをクライアントから削除して、新しいオブジェクトをサーバからダウンロードする。

【0069】20 例えば、図7の実施例においては、クライアント2のアプリケーションプログラム2-1におけるオブジェクト2-4Aが不要となったので、これを削除している。そして新たに必要になったオブジェクト1-4Aをサーバ1からクライアント2に対してダウンロードする。

【0070】これにより、次のことが可能になる。

【0071】(1) ソフトウェアをアップデートすることができる。例えば、ハードウェア制御ソフトにバグが見つかった場合に、そのオブジェクトを削除して、新しいオブジェクトに置き換える。家庭電化製品は、コンピュータの専門家ではない、一般消費者が利用するものなので、一部のコンピュータに見られるようなインストーラによるソフトウェアのアップデートは適切ではない。

【0072】(2) 30 製品のサイクルを長くすることができる。例えば、テレビジョン受像機は毎年のようにモデルチェンジが繰り返されるが、一般消費者は、テレビジョン受像機を毎年買い換るようなことはしない。しかし、本システムにより、ソフトウェアの問題はすべて最新の機能をユーザに提供できることになるので、ソフトウェアの機能拡張によるモデルチェンジをなくすことができる。これは、STBの場合にも当てはまる。

【0073】(3) 40 ユーザインタフェースの好みの変化に対応できる。例えば、初めてその装置を使い始めたユーザには、親切なメニュー形式を提供するが、ユーザがその装置の使い方を習熟するにつれて、直接所望の操作ができるようなダイレクト操作のユーザインタフェースに変更することができる。この場合も、両方の手順をクライアント側に持たせるのではなく、そのときのユーザの熟練度に応じたユーザインタフェースをクライアント側に持たせることができる。これにより、限りあるクライアントの資源を有効に利用することができる。

【0074】また、オブジェクトをダウンロードすれば、図8に示すようにクライアント2のオブジェクトを動的に拡張することができる。図8の実施例において50 は、クライアント2におけるアプリケーションプログラ

ム21-1のオブジェクト24-1Bに対して新たなサービスを受けられるようにするために、実行環境22-2が生成されている。そして実行環境22-1から必要なオブジェクト25-1A, 25-1Bが実行環境22-2にオブジェクト25-2A, 25-2Bとなるように、マイグレート(移転)する。さらに必要なその他のオブジェクト25-1C, 25-1Dも実行環境22-2にマイグレートされる。

【0075】そしてアプリケーションプログラム21-2には、アプリケーションプログラム21-1のオブジェクト24-1Bが、オブジェクト24-2Bとなるようマイグレートされる。

【0076】このようにして、例えば、実時間スケジューリングの拡張が必要になった場合には、その為の新しい実行環境をクライアントに生成し、必要なオブジェクトをその新しい環境に移動する。オブジェクトには何も変更を加える必要がなく、オブジェクトは実時間スケジューリングのサービスを受けられるようになる。

【0077】これにより次の効果が得られる。

【0078】(1) アプリケーションプログラムのオブジェクトに変更を加えないで、新しい機能に対処することが可能になるので、アプリケーションプログラムの寿命が長くなり、再利用可能性が増す。従来の手法では、アプリケーションプログラムにその実行環境に対する依存コードが含まれていたために、実行環境が変わることは、アプリケーションプログラムの書き換えを意味していた。

【0079】(2) 組み込み機器のアプリケーションプログラムの場合には、ユーザインタフェースをはじめとする機器の高機能制御ソフトの部分は、モデルが大きく変わらない限り再利用したい部分であり、また、既存のコードを利用して機能拡張するのみで開発期間を短縮したい部分であるが、この部分に、実行環境への依存コードが含まれていると、再利用のための作業は複雑になる。今までの手法は、この部分に対して何の戦略もなかったが、本方式によって、この作業は自動化、あるいは、最小化できる。

【0080】本システムは、具体的には次のように応用できる。

【0081】(1) アプリケーションの信頼性が低い場合に、メモリ保護機能をダウンロードすることによってシステム全体の停止を防ぐことができる。

【0082】(2) STBのベンダ毎に、サービスプロバイダは提供サービスを変えることができる。例えば、映画会社Aの映画を提供するサービスプロバイダは、その映画を受信するB社のSTBとC社のSTBとで、画像の質を変えることができる。

【0083】(3) STBに送られてくるA/Vデータの圧縮方式やユーザの画質の好みに応じて、システムの処理方式を変更する。例えば、MPEGデータの場合と

JPEGデータでは画質を調整するときの仕方が異なるので、システムの処理方式を変更する必要があるが、本システムでは、データ形式に従って動的に処理方式を選択できる。

【0084】このように、システムのほとんどの機能はサーバ1からダウンロードすることができるので、あらかじめクライアント2に様々な機能を持たせる必要が無く、最小限の機能を持たせればよい。図9に本システムにおけるクライアント2が有する最小限の機能を示す。

【0085】すなわち、クライアント2には、デバイスドライバのための実行環境22-1、システムオブジェクトのための実行環境22-2、実行環境の為の実行環境22-3を、最低限形成する。

【0086】(1) アプリケーションプログラム(ビデオ、ゲーム、ショッピング等)に適したクライアント環境を動的に構成する実施例を表している。図10では、ショッピングのアプリケーションプログラム11-2をサーバ1からダウンロードするために、ショッピング用の実行環境22-4がクライアント2-2に構成されている。また、ショッピングアプリケーションプログラムから映画アプリケーションプログラムに、クライアント2-2がアプリケーションプログラムを切り替えた場合には、映画アプリケーションプログラム21-3の実行環境22-3がクライアント2-2に構成され、サーバ1から映画アプリケーションプログラム11-1がダウンロードされる。

【0087】例えば、次のような処理が考えられる。

【0088】(1) ユーザが映画を選択しているときこの時には、所望の映画を選択することができるよう、ナビゲーションアプリケーション11-3が、例えばクライアント2-1の実行環境22-1にダウンロードされ、それに対する必要な環境として、ウィンドウ管理、ユーザからの入力管理等のオブジェクト15-3がオブジェクト25-1としてダウンロードされる。

【0089】(2) ユーザが映画を鑑賞しているときこの時には、サーバ1-1の実行処理12-1から、ビデオストリーム管理、データ先読みバッファ管理、VCR機能等のオブジェクト15-1が、例えばクライアント2-2の実行環境22-3にオブジェクト25-3と

してダウンロードされる。

【0090】以上のように、クライアントの実行環境をダウンロードにより実現するために、本システムでは図11に示すフィーチャストラクチャ (feature structure) を導入する。オブジェクトがサーバ1からクライアント2にダウンロードされる時、このfeature structureが検査されて、必要な実行環境がダウンロード先に構成される。

【0091】すなわち、サーバ1は、クライアント2との間において、第1のフェーズ (ネゴシエーションフェーズ) で、サーバ1側のメタオブジェクト空間とクライアント2側のメタオブジェクト空間との間におけるオブジェクトマイグレーションの可能性に関するネゴシエーションを行う。そして第2のフェーズ (移動フェーズ)において、実際にオブジェクトの転送を行う。

【0092】オブジェクトマイグレーションはメタレベルのプロセスであり、オブジェクトの内部情報と、必要ならばそれに関連したオブジェクトによって使用中の計算資源を転送するものである。オブジェクトの内部情報は、ディスクリプタ (descriptor) と呼ばれるメタレベルオブジェクトによって表現されている。ディスクリプタは、実際には、オブジェクトを管理しているメタオブジェクトの名前を保持している。一般的なディスクリプタは、オブジェクトのメモリセグメントを管理しているメタオブジェクトの名前、2つ以上のオブジェクトの実行制御をしているメタオブジェクト (スケジューラ) の名前、オブジェクトの名前付けを管理しているメタオブジェクトの名前、等を保持している。

【0093】第1のフェーズ (ネゴシエーションフェーズ)においては、オブジェクトの移動の可能性が検査される。すなわち、メタオブジェクト空間によっては、オブジェクトマイグレーションが望ましくない場合がある。例えば、デバイスドライバを管理しているメタオブジェクト空間が、オブジェクト (デバイスドライバ) を移動する場合には、クライアント2側にハードウェアデバイスが実際に存在していなければ、その移動を行っても意味が無くなってしまう。また仮想記憶管理機構を利用して、オブジェクトのメモリセグメントを管理しているメタオブジェクトも、移動先に仮想記憶管理機構が存在しなければ、移動を行ったとしてもメモリセグメントを管理することができない。そこで、マイグレーションプロトコルにおいては、次のメソッドが用意されている。

【0094】`Feature* Descriptor::CanSpeak (Feature* pFeature)`

【0095】このメソッドは、クライアント2側のメタオブジェクト空間内の`Descriptor`に対して、`CanSpeak`オペレーションを実行する。

【0096】この時、フィーチャストラクチャが引数と

して渡され、結果としてクライアント2は、サーバ1に対して自分自身 (クライアント2) が受け入れ可能なフィーチャストラクチャを返す。サーバ1側においては、クライアント2側から返されたフィーチャストラクチャを検査することで、そのメタオブジェクト空間の互換性を知ることができる。

【0097】互換性は、完全互換、半互換、および非互換の3つのカテゴリに分類される。

【0098】完全互換は、オブジェクトが移動後も完全に実行を続けることができる場合を意味する。半互換は、移動後のオブジェクトの実行には、ある制限が加えられることを意味する。そして、非互換は、オブジェクトは移動後には実行を続けることができないことを意味する。

【0099】非互換の場合には、オブジェクトマイグレーションは行われない。半互換の場合にはマイグレーションを行うか、行わないかをユーザが判断する。実際には、例外がユーザに返されるので、例外処理ルーチンによって、マイグレーションの判断がなされる。完全互換の場合、または半互換の場合であって、オブジェクトマイグレーションが行われるときには、先に返されたフィーチャストラクチャの内容に従って、マイグレーションが行われる。

【0100】なお、ネゴシエーションフェーズに先だって、次のオペレーションによって空の`Descriptor`がクライアント2側のメタオブジェクト空間に生成される。

【0101】`Descriptor::Descriptor ()`

【0102】先の`CanSpeak`メソッドは、この`Descriptor`に対して送られる。この時、フィーチャストラクチャの情報を元に必要なメタオブジェクトの生成、参照の生成、必要情報の登録が行われる。

【0103】第2のフェーズにおけるメタレベルでのプロセスは、転送オブジェクトに対応したメタオブジェクトの移動、あるいは転送になる。ここで、メタオブジェクトの移動とは、そのメタオブジェクトがクライアント2側のメタオブジェクト空間に入ること、すなわち、ディスクリプタ (`Descriptor`) から参照されるようになることを意味し、また、メタオブジェクトの転送とは、クライアント2側のメタオブジェクト空間にあるメタオブジェクト (これは`Descriptor`から参照されている) に対して、メタオブジェクト内のデータをメッセージとして送ることを意味する。

【0104】メタオブジェクトの移動および転送に関する実際のオペレーションが、ネゴシエーションフェーズによって得られたフィーチャストラクチャを利用して、この第2のフェーズ (移動フェーズ) において実行される。

【0105】移動フェーズにおける実際のメタオブジェ

クトの移動および転送は、次のメソッドによって起動される。

【0106】Descriptor & Descriptor::operator=(Descriptor & source)

【0107】Descriptorクラスは抽象クラスであり、メタオブジェクトの移動、転送に関する共通のプロトコルを定義している。sourceで参照されるdescriptorの内容が、このディスクリプタにコピーされる。

【0108】実際の手続きは、Descriptorクラスのサブクラスとして定義される。

【0109】次のメソッドは、メタオブジェクトの転送に関するメソッドである。これらのプロトコルは主にmigrator(メタオブジェクト空間内に含まれる、オブジェクトマイグレーションを担当するメタオブジェクト)によって使用される。

・CanonicalContext & Context::asCanonical()

機械依存のContext構造体を、機械非依存の形式に変換する。このプロトコルは、feature structureによってContextの直接変換が不可能である、と示されたときに実行される。

・Context & Context::operator=(Context & source)

・Context & Context::operator=(CanonicalContext & source)

現在(thisが参照する)Contextをsourceが参照するContextで初期化する。

・CanonicalSegment & Segment::asCanonical()

機械依存のSegment構造体を、機械非依存の形式に変換する。このプロトコルは、feature structureによってContextの直接変換が不可能である、と示されたときに実行される。

・Segment & Segment::operator=(Segment & source)

・Segment & Segment::operator=(CanonicalSegment & source)

現在のthis(が参照する)Segmentをsourceが参照するSegmentで初期化し、必要なメモリ領域をコピーする。

【0110】図11は、フィーチャストラクチャの構造を表している。同図に示すように、object descriptionとenvironment descriptionのポインタが、エントリーに記述されている。

【0111】object descriptionの

ポインタで指示されているストラクチャには、object nameが記述され、さらに、environment descriptionのポインタで指示されているストラクチャと同一のストラクチャのポインタが記述されている。そしてさらに、このオブジェクトの資源要求(resource requirement of this object)が記述されている。

【0112】また、environment descriptionのポインタで指示されているストラクチャには、environment nameが記述され、さらに、そのクライアントのハードウェアの資源情報(resource information of client hardware)、この実行環境の資源要求(resource requirement of this environment)、およびこの実行環境を構成するメタオブジェクトのリスト(list of metaobjects constituting environment)が記述されている。

【0113】フィーチャストラクチャの内容の具体例をあげると、次のようにになる。

【0114】(1) オブジェクトに関する情報

実時間性

必要プロセッサ量

【0115】(2) メタオブジェクトに関する情報

ハードウェアメタオブジェクト

*プロセッサの種類

*データ表現形式

セグメントメタオブジェクト

30 *大きさ

*拡大、縮小可能性

*管理方針

*レイアウト

コンテキストメタオブジェクト

*レジスタ情報

*一時変数

*プロセッサ状態

メイラメタオブジェクト

*メッセージキュー長

40 *未処理メッセージ数

*外部メイラの必要性

*メッセージの転送方式

外部メイラメタオブジェクト

*メッセージキュー長

*未処理メッセージ数

*プロトコル

スケジューラメタオブジェクト

*オブジェクト状態

*スケジューリング方針

50 依存管理メタオブジェクト

*保有する外部名数

【0116】上述したように、各クライアントが異なるOSを有するような場合、このフィーチャストラクチャから各クライアントのOSが判定され、サーバは、そのOSに対応するオブジェクトをダウンロードさせることになる。

【0117】図12は、本発明のデータ処理システムを応用したApertos（商標）システムの構成例を示している。このApertosシステムにおいては、Apertos Micro Virtual Machine (MVM) 31a (第1の実行手段) は、Apertos Micro Kernel (MK) 31b (第2の実行手段)とともにあり、Apertosシステムの核31を構成している。核31を構成するMVM 31aは後述する中間コード (I-code) を解釈実行するが、必要に応じて、MK 31bの機能を用いてpersonalitiyオブジェクト（システムオブジェクト）を呼び出すようになされている。

【0118】図12に示した核31以外の部分は、上述した方法により、例えば、サーバ1よりダウンロードすることができる。

【0119】このシステムは、必要に応じてI-codeをnative code（バイナリコード、マシンコード）に動的にコンパイルする。また、予めnative codeにコンパイルされているオブジェクトを実行するが、その場合、MK 31bの機能によってPersonalitiyオブジェクト33（バイナリコード生成手段）が呼び出され、アプリケーション（Applications）35にサービスを提供する。

【0120】図12に示した核31を構成するMVM 31aおよびMK 31bは、その外側がデバイスドライバオブジェクト（Device drivers）32とPersonalitiyオブジェクト（Personalitiy component objects）33によって取り囲まれており、さらに外側には、アプリケーションプログラミングのためのクラスシステム（Class Libraries）34が用意されており、さらにその外側にはアプリケーション35が用意されている。

【0121】Personalitiyオブジェクト33の層により、Apertosシステムは様々なOS（オペレーティングシステム）やVirtual Machine（仮想計算機）を提供することができる。例えば、javaプログラムの実行は、javaプログラム用のPersonalitiyオブジェクトによって、javaプログラムがコンパイルされて得られた中間コードであるjava bytecodeが実行されることにより行われる。

【0122】Apertosシステムは、高度のportabilityを実現するために、プログラムをI-

code（中間コード）にコンパイルし、オブジェクトのメソッドを管理している。しかしながら、I-codeは、解釈実行（プログラムを解釈しながら実行すること）を前提に設計されているのではなく、動的コンパイラによってnative codeにコンパイルするよう設計されている。しかし、様々な制約により、動的コンパイルが困難である場合、MVM31aがI-codeを解釈実行する。

【0123】しかしながら、ほとんどの場合、I-codeは、native codeにコンパイルされ、システムを構成するCPUにより直接実行される。従って、Virtual Machineによる実行に伴って、実時間性が欠如したり、処理スピードが犠牲になるということはほとんどない。

【0124】上記I-codeは、高度のInter-Operabilityを実現するため、図22を参照して後述するように、十分に抽象度の高い2つの命令セット（OP_M, OP_R）から構成されており、そのセマンティクス（意味構造）は、MK 31bのインターフェースと強く関連づけられている。即ち、図12に示したApertosシステムの構造に強く影響を受けた命令セットとされている。これにより、Apertosシステムにおいては、native codeを前提としているながら、高度のPortabilityとInter-Operabilityを実現することができる。

【0125】次に、MVM31aとMK31bの仕様について説明する。最初に、MVM31aが仮定しているデータ構造体と、I-codeフォーマットを規定する。

【0126】図13は、図12に示したMVM31aとMK31bの論理構造を示している。基本的に両者の論理構造は同一であり、active context 41, message frame 42、およびexecution engine 43により構成されているが、MVM31aは、I-codeによる実行をサポートし、MK31bはnative codeによる実行をサポートしている。

【0127】図13において、active context 41は、現在実行中のContext（図14を参照して後述する）を指しており、message frame 42は、核31を構成するMVM31aおよびMK31bに対するメッセージ本体を指している。

【0128】MK31bの場合、メッセージ本体の存在する場所は実装方法に依存し、スタッカフレームとしてメモリに割り当てられる場合もあるし、ヒープ（heap）に割り当てられる場合もある。また、いくつかのCPUのレジスタが割り当てられる場合もある。一方、MVM31aの場合、メッセージフレームは命令コードに続くオペランドを指すことになる。実装によっては、これら以外に内部レジスタ等を必要とするかもしれない

が、それらのレジスタは本仕様とは独立のものである。

【0129】図13において、`execution engine` 43は、`I-code` および `native code` を実行する。また、MVM31aの`execution engine` 43には、`primitive objects` 44が含まれており、これは、図18を参照して後述するプリミティブオブジェクトを処理するコードである。

【0130】図14は、`Context` と `Descriptor` の論理構造を示している。プログラムの1つの実行状態を示す`Context` 構造体51は、`object`、`class`、`method`、`meta`などのフィールドを有し、このうち、`method`には、`icode` と `minfo` のフィールドがさらに設けられている。この`Context` 構造体51は、MVM31aの状態を保持し、CPUのレジスタに相当し、メモリ管理やプログラム間の通信方法などとは完全に独立している。

【0131】`Context` 構造体51は、`Context` プリミティブオブジェクトであり、図16を参照して後述するように、その各フィールドは所定の情報にリンクしている。また、`Context` 構造体51は、核31(MVM31a および MK31b)の実装に深く依存しているが、図14においては、主に依存しない部分を示している。

【0132】この`Context` 構造体51において重要なフィールドは、`meta` フィールドであり、`Descriptor` 構造体52を指している。`Descriptor` 構造体52のエントリは、`#tag`、`context`、および `selector` の3つの組みであり、この`Descriptor` 構造体52によって、`Personality` オブジェクト33のAPI(アプリケーションプログラミングインターフェース)が決定される。`#tag` はAPIの名称を、`context` と `selector` でAPIのアドレスをそれぞれ表している。

【0133】図15は、MVM31aの全体構造を示している。基本的に、全ての必要な情報は、`Context` 構造体51からリンクを辿ることにより参照することができるようになされている。即ち、`Context` 構造体51が、`object` 61にリンクされ、この`object` 61は、`object` 61に対応するクラスへのリンク(`class pointer`)とインスタンス領域(`object dependent field`)より構成される。インスタンス領域にどのような情報が格納されるか、あるいはそのレイアウトは、オブジェクトの実装に依存する。

【0134】`class` 62は、主としてメソッドを保持する。`class` 62はその名称(`class name`)、その実装に依存する部分(`class dependent field`)、および `I-method`

構造体63へのリンクテーブル(`method table`)により構成される。`I-method` 構造体63は、`Block` プリミティブオブジェクトであり、ヘッダ(Header)、`I-code` 部、および変数テーブル(`Variable table`)からなる。`magic#` は、MVM31aの管理番号(ID)である。基本的に、`I-code` 命令のオペランドは、この変数テーブルエントリを介して目的オブジェクトを参照する。

【0135】図15において、グレーの部分(`Context` 51、`I-method` 63)はMVM31aに依存する部分であり、MVM31aが `I-code` を解釈実行する場合に必要とする構造体である。

【0136】図16は、`Context` 構造体51を中心に、そこからリンクされている構造体を図示したものである。`Context` 構造体51のオブジェクトフィールドは、`object` 61を指し、`class` フィールドは、`class` 62の`class dependent field`を指している。また、`method` フィールドは `I-method` 63を指しており、`method` フィールドの`icode` は、`I-method` 63の`I-code` を指している。`icode` は、プログラムカウンタに相当し、実行中のプログラムを指す。`method` フィールドの`vtable` は、`I-method` 63の`variable table`を指している。

【0137】`temporary` フィールドは、一時データ保存領域を指し、`meta` フィールドは、上述したように、`Descriptor` 52を指している。また、オブジェクト61の`class pointer` は、`class` 62を指し、`class` 62の`method table` は、`I-method` 63を指している。

【0138】`variable table` エントリは、`type` と `value` の組みであり、`value` は `type` に依存する。

【0139】MVM31aは、図17に示したタイプを処理するようになされており、`type` が `T_PRIMITIVE` のとき、`value` 欄はプリミティブオブジェクトを参照する。この場合、`value` 欄には、例えば、`<P_INTEGER, immediate>`、`<P_STRING, address to heap>` といった、図18に示すようなく `P_CLASS`、`P_BODY` の組が格納される。図18は、プリミティブオブジェクトの一覧と、そのインターフェース名を示している。図19乃至図21は、図18に示したプリミティブオブジェクトのインターフェースを、`Interface Definition Language (IDL)` 法により記述した例を示している。このIDL法について、「CORBA V2.0, July 1999

5, P 3-1 乃至 3-36」に開示されている。

【0140】また、図17において、typeがT_P_OINTERのとき、オブジェクトを参照するためのIDがvalue欄に格納される。このIDは、その位置が独立な、システムで唯一の値であり、Personalityオブジェクト33によって、そのIDに対応するオブジェクトの位置が特定される。

【0141】図22は、MVM31aが解釈実行する2つの命令セットを示している。命令セットOP_Mは、図12において、外側から内側に入る命令であり、命令セットOP_Rは、内側から外側に戻る命令である。

【0142】即ち、命令セットOP_Mにより、その第1オペランドで示されたオペレーションが実行される。このオペレーションは、Personalityオブジェクト33によって処理されるか、またはプリミティブオブジェクトによって処理される。コンパイラは、より効率的にアプリケーション35を実行するために、いくつかの処理をプリミティブオブジェクトによって処理するように、I_codeを生成する。これにより、例えば、Integerの算術演算は、図18の先頭のプリミティブオブジェクトによって処理される。

【0143】次に、MK31bのインターフェースを規定する。図23は、マイクロカーネル(MK)31bのインターフェースを示している。MVM31aとMK31bの論理構造は、図13に示したように同様であり、MK31bは、図22に示した命令セットOP_MおよびOP_Rをシステムコールとして処理する。

【0144】上述したような構成のApertosシステムを、例えば図4に示したクライアント2(2-1, 2-2)に適用することができる。そして、図12に示した核31以外のもののうち、任意のものをサーバ1からダウンロードすることにより、所定のアプリケーションを実行するための最適な実行環境をクライアント2上に構築することができる。

【0145】上述したように、ここでダウンロードするオブジェクトとしては次のものが考えられる。

【0146】(1) すべてのアプリケーションプログラム

【0147】(2) クライアントが備えるハードウェア資源を制御するためのデバイスドライバ群(例えば、MP EGドライバ、ATMドライバ、画像制御ドライバ等)

【0148】(3) アプリケーションプログラムに対してシステムサービスを提供するオブジェクト群(personalityオブジェクト)(例えば、VCRコマンド管理、ストリーム管理、実時間スケジューラ、メモリ管理、ウインドウ管理、ダウンロード制御、通信プロトコル管理、実行管理等)

【0149】これらを組み合わせて、アプリケーションプログラムに対して最適な実行環境をクライアント上に

構築することができる。

【0150】例えば、javaプログラムを実行させたい場合、java用のpersonalityオブジェクト33と、javaプログラム(アプリケーション)35をサーバ1よりダウンロードする。このpersonalityオブジェクト33により、Apertosシステムは、Java Virtual Machineを提供することができる。ダウンロードされたjavaプログラムは、コンパイラにより一旦java bytecodeと呼ばれる中間コードにコンパイルされ、上記Java Virtual Machineにより実行される。あるいは、上述したように、native codeにコンパイルした後、実行するようにすることも可能である。

【0151】以上の実施例においては、サーバからクライアントに所定のオブジェクトをダウンロードするようになしたが、所定のクライアントから、所定のサーバにオブジェクトをダウンロードしたり、あるいはサーバ同士、クライアント同士でオブジェクトをダウンロードする場合にも、本発明を適用することが可能である。

【0152】また、上記実施例においては、I_codeを2つの命令により構成するようにしたが、これに限定されるものではない。

【0153】

【発明の効果】以上の如く請求項1記載のデータ処理方法および請求項9に記載のデータ処理装置によれば、クライアントがダウンロードするアプリケーションプログラムの実行環境を有するか否かを検査し、その検査結果に対応して、アプリケーションプログラムをクライアントにダウンロードするようにしたので、クライアントの構成を簡略化し、低コスト化するとともに、低コストでアプリケーションプログラムを提供することが可能となる。

【0154】請求項10に記載のデータ処理装置によれば、サーバに対してダウンロードするアプリケーションプログラムを実行環境に関する告知を行い、その告知に対応して、サーバからアプリケーションプログラムをダウンロードするようにしたので、構成が簡単で低コストの装置を実現することができる。また、低コストでアプリケーションプログラムを提供することができる。

【0155】請求項11に記載のデータ処理装置、および請求項15に記載のデータ処理方法によれば、中間コードに変換されたアプリケーションプログラムを解釈し、実行するか、または、中間コードを動的にコンパイルし、生成されたバイナリコードを実行することで、アプリケーションプログラムを実行するようにしたので、動的コンパイルが困難な場合、中間コードを逐次解釈し、実行することができる。また、中間コードを簡単な構成とすることにより、ポータブルなアプリケーションを構築することができる。

【図面の簡単な説明】

【図1】本発明のデータ処理方法を適用するシステムの構成例を示す図である。

【図2】アプリケーションプログラムの構成を示す図である。

【図3】並行オブジェクトの構成を示す図である。

【図4】サーバから複数ベンダのクライアントに対するオブジェクトのダウンロードを説明する図である。

【図5】インクリメンタルダウンロードを説明する図である。

【図6】メタ標準を説明する図である。

【図7】オブジェクトの動的変更を説明する図である。

【図8】オブジェクトの動的拡張を説明する図である。

【図9】クライアントの最小機能を説明する図である。

【図10】アプリケーションに適したクライアント環境の構築とその動的再構成を説明する図である。

【図11】フィーチャストラクチャの構成を説明する図である。

【図12】本発明のデータ処理装置を応用したApertosシステムの構成例を示す図である。

【図13】MVMとMKの論理構造を示す図である。

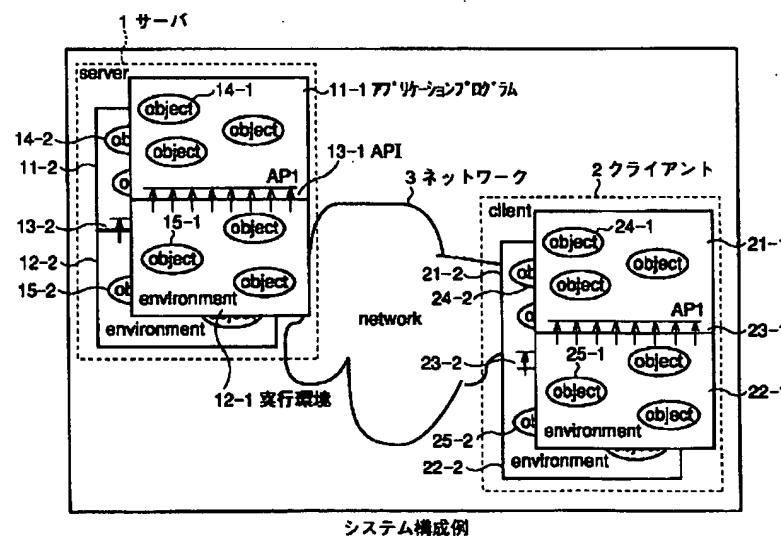
【図14】ContextとDescriptorの論理構造を示す図である。

【図15】MVMの全体構造を示す図である。

【図16】Contextとその回りのデータ構造体を示す図である。

【図17】Variable tableエントリのt

【図1】



ypeを示す図である。

【図18】プリミティブオブジェクトの一覧とそのインターフェース名を示す図である。

【図19】プリミティブオブジェクトのインターフェースを示す図である。

【図20】プリミティブオブジェクトのインターフェースを示す図である。

【図21】プリミティブオブジェクトのインターフェースを示す図である。

【図22】I-code命令セットを示す図である。

【図23】MKのインターフェースを示す図である。

【符号の説明】

1 サーバ, 2 クライアント, 3 ネットワーク, 1

1 アプリケーションプログラム, 12 実行環境, 1

3 アプリケーションプログラムインターフェース, 1

4, 15 オブジェクト, 21 アプリケーションプログラ

ム, 22 実行環境, 23 アプリケーションプログラ

ムインターフェース, 31 核, 31a MVM, 31

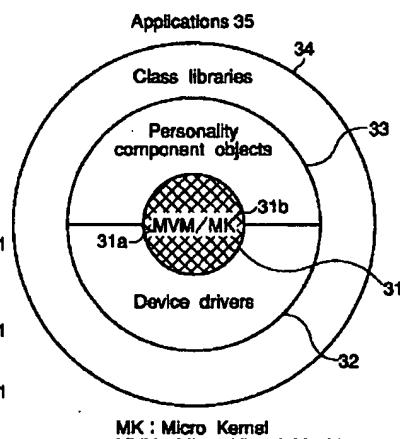
b MK, 32 Device drivers, 33

20 Personality, component object, 34 Class libraries, 3

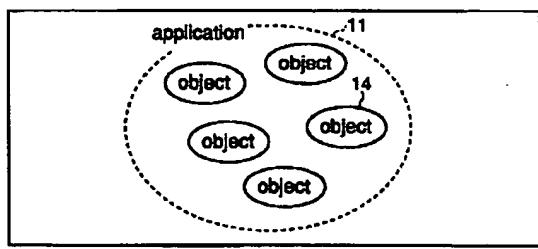
5 Applications, 41 active context, 42 message frame,

43 execution engine, 44 primitive objects, 51 context, 52 Descriptor, 61 object, 62 class, 63 I-method

【図12】

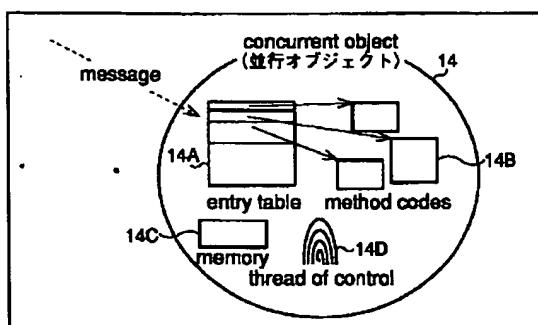


【図2】



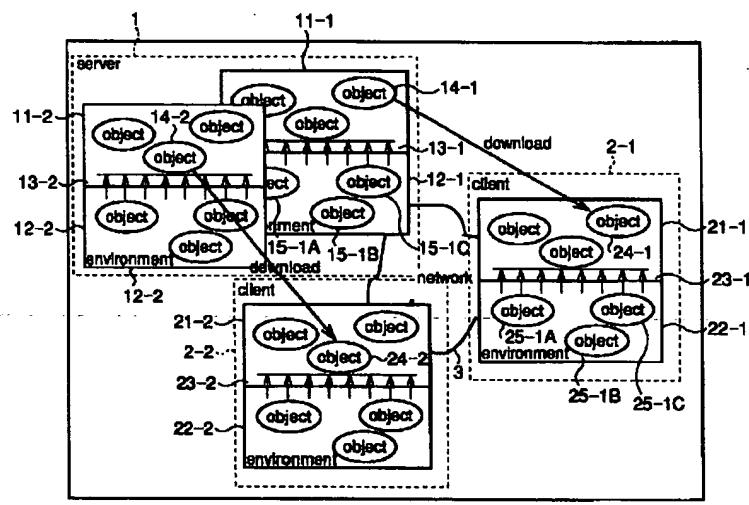
アプリケーションの構成

【図3】



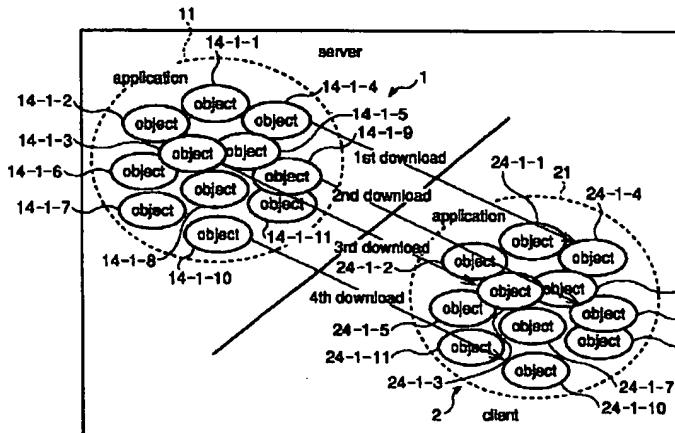
並行オブジェクトの構造

【図4】



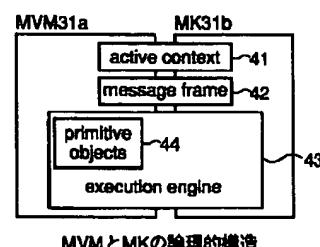
複数ペンドのクライアントにサーバからオブジェクトをダウンロードする方法

【図5】

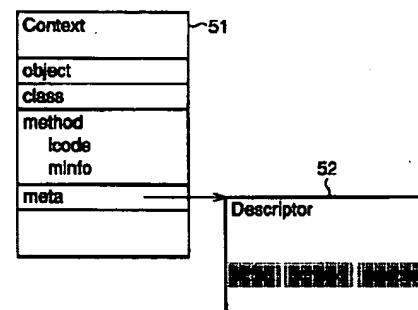


インクリメンタルダウンローディング

【図13】



【図14】



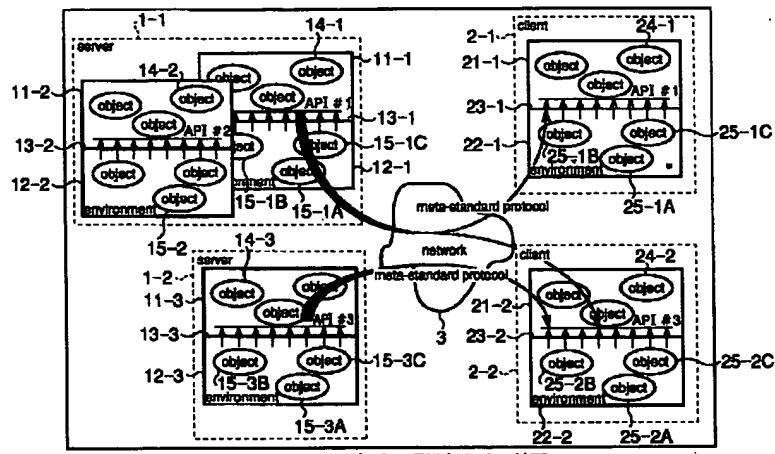
ContextとDescriptorの論理構造

【図17】

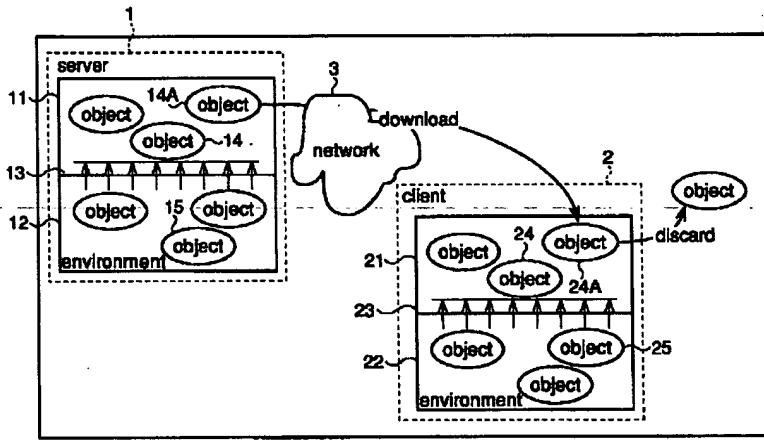
type	description
T_INVALID	invalid entry
T_PRIMITIVE	The value field denotes a primitive object, which is a dependent object on MVM
T_POINTER	The value field contains a pointer to an object

variable tableエントリのtype

【図6】

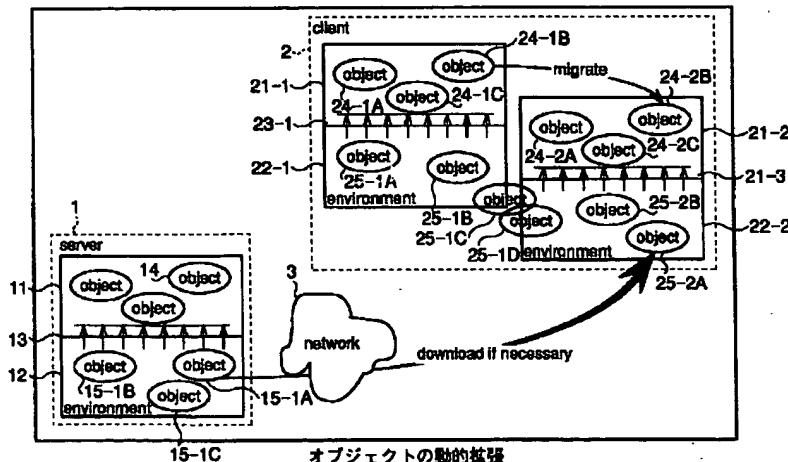


【図7】



オブジェクトの動的変更

【図8】



オブジェクトの動的拡張

【図19】

```

19-1
struct Message {
    long length;
    any body [];
};

exception outrange (long position);
exception overflow ();
exception undef (int primitive; int selector);
exception zerodiv ();

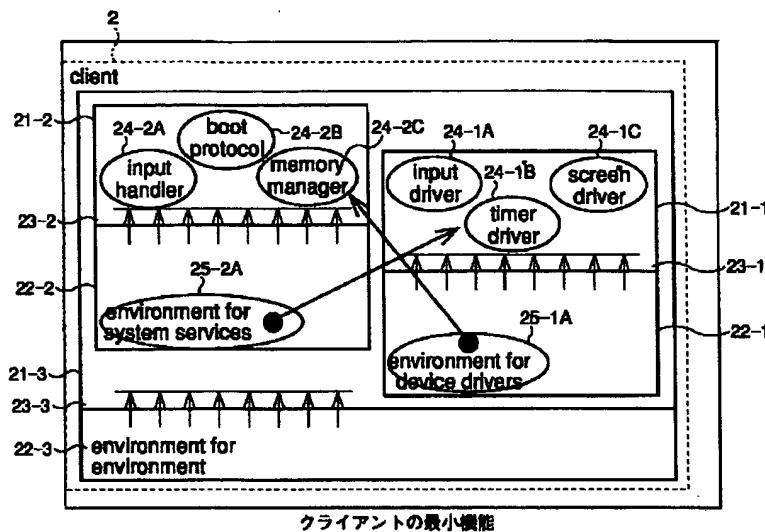
interface Integer {
    Integer operator + (in Integer value)
        raise (overflow);
    Integer operator - (in Integer value)
        raise (overflow);
    Integer operator * (in Integer value);
    Integer operator / (in Integer value)
        raise (zerodiv);
    Integer remainder (in Integer value);
    Integer and (in Integer value);
    Integer or (in Integer value);
    Integer xor (in Integer value);
    Integer not (void);
    Integer operator = (in Float value);
    Integer operator = (in DoubleFloat value);
    Boolean operator < (in Integer value);
    Boolean operator <= (in Integer value);
    Boolean operator == (in Integer value);
};

interface Float {
    Float operator + (in Float value)
        raise (overflow, underflow);
    Float operator - (in Float value)
        raise (overflow, underflow);
    Float operator * (in Float value)
        raise (overflow, underflow);
    Float operator / (in Float value)
        raise (overflow, underflow, zerodiv);
    Float operator = (in Integer value);
    Float operator = (in DoubleFloat value);
    raise (overflow);
    Boolean operator < (in Float value);
    Boolean operator <= (in Float value);
    Boolean operator == (in Float value);
};

```

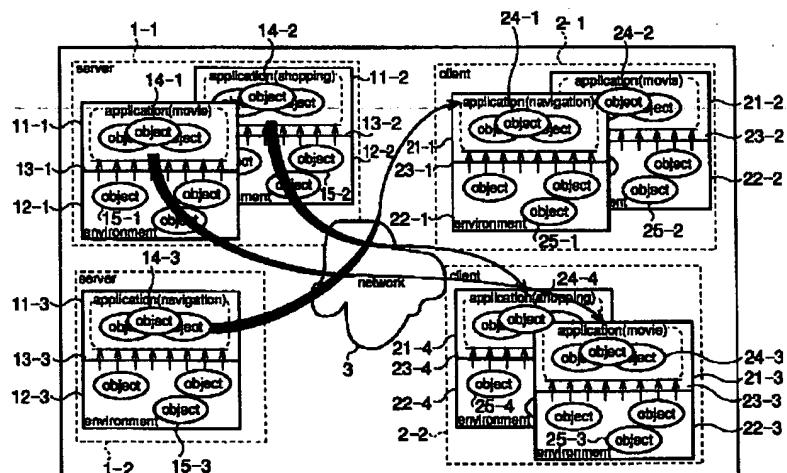
プリミティブオブジェクトインターフェース

【図 9】



クライアントの最小機能

【図 10】



アプリケーションに適したクライアント環境の構築と、その動的再構成

【図 2 2】

name	description
OP_M	execute the operation specified by the first argument. This operation is processed either by personality or by a primitive object.(operands:operation,number of operands,...)
OP_R	give control back to the object when the operation requested by OP_M returns (operands:target,return arguments)

I-code 命令セット

【図 2 3】

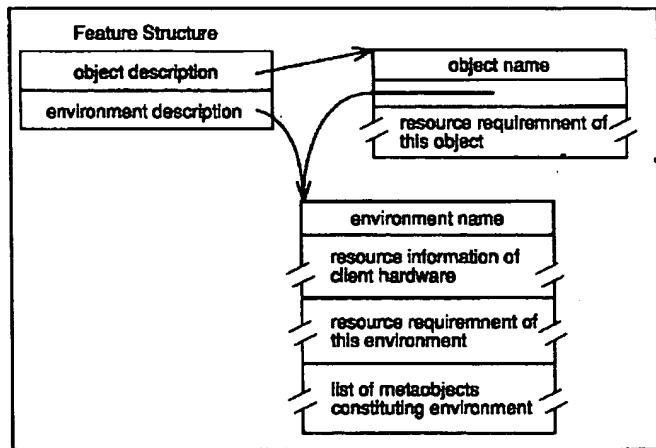
```

Interface MetaCore {
    mcError M (in long method, in Message msg);
    mcError R (in CName obj, in long method, in Message msg);
}

```

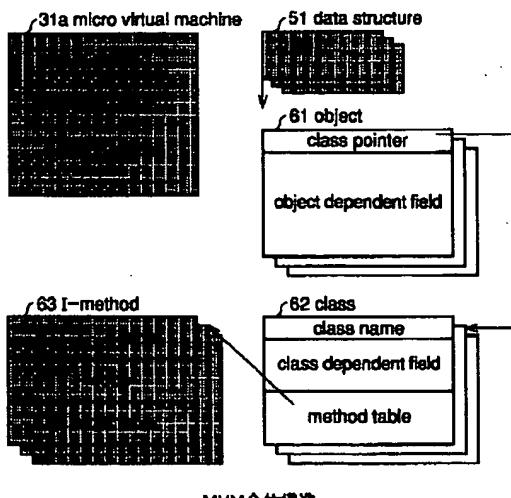
Micro Kernelのインターフェース

【図1-1】

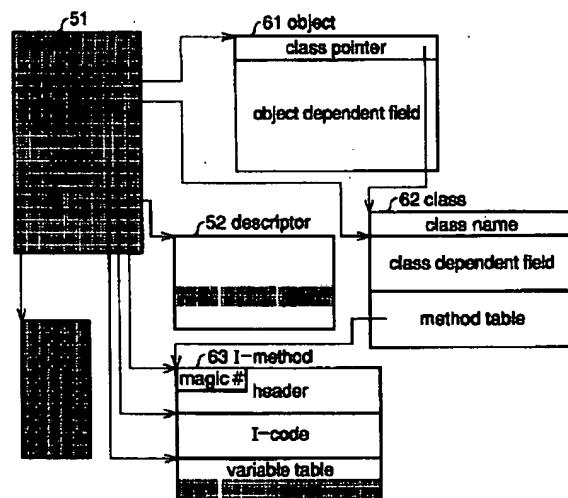


Feature Structureの構造

【図1-5】



【図1-6】



【図18】

class	P_CLASS	P_BODY	description
Integer	P_INTEGER	immediate	Integer number
Float	P_FLOAT	immediate	IEEE 754 32 bits floating point number
Double Float	P_DFLOAT	immediate	IEEE 754 64 bits floating point number
Boolean	P_BOOLEAN	immediate	TRUE or FALSE
String	P_STRING	address to heap	sequence of UNICODE characters
Array	P_ARRAY	address to heap	sequence of data of the same type
Context	P_CONTEXT	CName	representation of object execution
Jump	P_JUMP	I-code index	target address to which execution resumes
Pointer	P_POINTER	address to memory	address
Assign	P_ASSIGN	n/a	duplication of a variable entry
Mailer	P_MAILER	n/a	Invocation of a method
Undef	P_UNDEF	n/a	undefined primitive object

プリミティブオブジェクト

【図21】

19-3

```
Interface Pointer {
    void put (in Address pos, in byte value)
    exception outrange;
    byte get (in Address pos)
    exception outrange;
    void put (in Address pos, in word value)
    exception outrange;
    word get (in Address pos)
    exception outrange;
    void put (in Address pos, in long value)
    exception outrange;
    long get (in Address pos)
    exception outrange;
    void put (in Address pos, in longlong value)
    exception outrange;
    longlong get (in Address pos)
    exception outrange;
};

Interface Assign {
    void operator = (in int position1, in int position2);
};

Interface Mailer {
    void call (in ID target, in Integer select, in Message msg);
    void reply ();
};

Interface undef
};
```

プリミティブオブジェクトインターフェース(続き)

【図20】

```
19-2
Interface DoubleFloat {
    DoubleFloat operator + (in DoubleFloat value)
    raise (overflow, underflow);
    DoubleFloat operator - (in DoubleFloat value)
    raise (overflow, underflow);
    DoubleFloat operator * (in DoubleFloat value)
    raise (overflow, underflow);
    DoubleFloat operator / (in DoubleFloat value)
    raise (overflow, underflow,zerodiv);
    DoubleFloat operator = (in Integer value)
    raise ();
    DoubleFloat operator = (in Float value)
    raise ();
    Boolean operator < (in DoubleFloat value)
    raise ();
    Boolean operator <= (in DoubleFloat value)
    raise ();
    Boolean operator == (in DoubleFloat value)
    raise ();
};

Interface Boolean {
    Boolean not (void);
};

Interface String {
    String operator + (in String string);
    String substr (in Integer position, in Integer length);
    raise (outrange);
    Integer length (void);
};

Interface Array {
    void put (in Integer Index, in any value);
    raise (outrange);
    any get (in Integer Index)
    raise (outrange);
    Integer length (void);
};

Interface Context {
    ...accessors...
};

Interface Jump {
    void eval (void)
    raise (outrange);
    void evalT (Boolean cond)
    raise (outrange);
    void evalF (Boolean condition)
    raise (outrange);
};

プリミティブオブジェクトインターフェース(続き)
```

プリミティブオブジェクトインターフェース(続き)